Soundboard Project

1) Architecture of the project:

The project is structured with a menu containing two views: the sampler view and the sound library view.

The sampler view offers a sampler with pads and when clicking on a pad a sound is played. A long click opens a pad editing view.

In this view the user can choose what to do:

- Crop the pad sound
- Change the source of the pad

If the user chooses to crop the pad sound, they are returned to a crop view. This view allows the user to change the start and end of the sound played.

If the user chooses to change the source of the pad they are returned to a source selection view. This view allows the user to choose from 3 sources:

- Choosing a new sound from the local library
- Recording a sound with the microphone
- Search for a sound on the freesound API

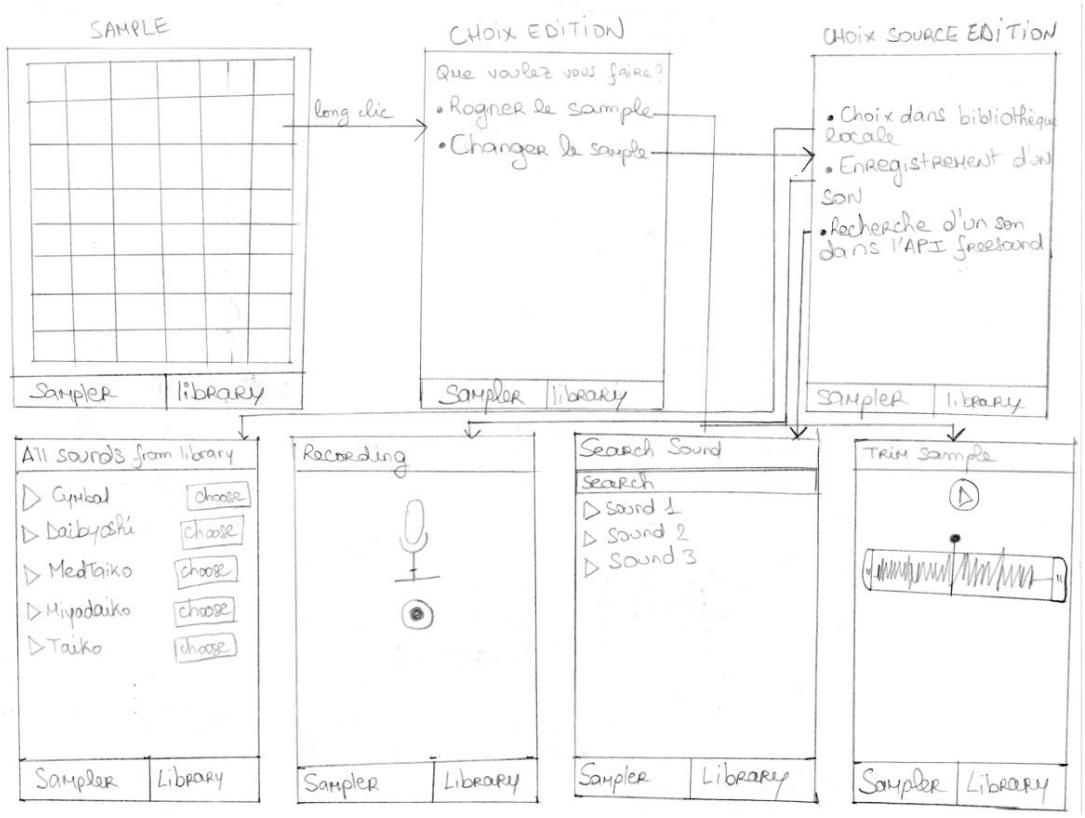
The "choose a new sound from the local library" view returns the list of sounds in the local library with a button for each one to choose the desired item.

The "Record a sound with the microphone" view returns a view that allows the user to record a sound with the microphone. Once the sound has been recorded, the user gives it a name and a description and can add it to the sound library.

The "search for a sound on the freesound API" view allows the user to search for a sound by keyword from the API. The user enters a keyword in the search bar and is presented with a list of sounds from the API.

The "library" navigation view allows the user to manage the sounds in the local library. He can launch sounds and delete them from the library.

Here is a graphic representing the architecture of the project.



2) Sampler

This view shows a sampler with pads containing different sounds using a flatlist. Each short press on a pad starts the sound. To store the sounds I used the redux store with a reducer to manage the complete library and a reducer to manage the pads.

The reducer for the library contains the sounds in the library, actions to add, edit or delete sounds. The reducer for the pads allows to initialize the pads and contains a changeSource action to change the source of the pad, i.e. to assign it a "sampleId" corresponding to an id of a sound of the library. This reducer also contains a selector to assign a url to a pad. These two reducers are then reused in the editing views.

3) Sample editing :

- Trimming the existing sample

This view links to a "trimmer" which allows you to choose the duration of the sound played by moving the edges of the trimmer. I use the edit action of the library reducer to pass the new edited sound with the start and end time of the sound.

- From the library sounds

This view returns a list of sounds from the local library using a flatlist. A choose button in front of each item allows to choose the new pad sound. For this I use the "changeSource" action of the reducer for the pad.

- From the microphone

This view allows you to use the phone's microphone thanks to the Expo Audio library using the Audio.Recording() function. Once the sound has been recorded the user assigns it a name and description and can add it to the sound library.

- From a search on the Freesound API.

A search bar allows the user to enter a keyword to search for a sound in the Freesound API. A list of results is presented and a button is used to select a sound. I use here the edit action of the library reducer to assign a new url to the pad.

4) Persistence

To be able to keep the data of the store I use redux-persist. Indeed I use the persistStore function to which I pass the store. This function is put in a variable which will be used by the PersistGate of App.js.